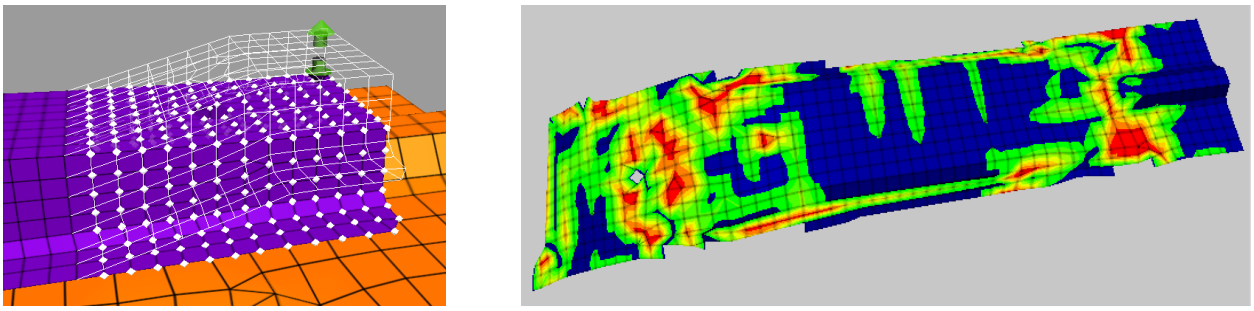
INTUITIVE, INTERACTIVE, AND ROBUST MODIFICATION AND OPTIMIZATION OF FINITE ELEMENT MODELS

Katrin Bidmon Dirc Rose Thomas Ertl

Visualization and Interactive Systems Group, University of Stuttgart
e-mail: (bidmon|rose|ertl)@vis.uni-stuttgart.de
web: http://www.vis.uni-stuttgart.de/



(a) (b)

Figure 1: Tools for interactive editing: (a) interactive stretching and deformation of a car component; (b) 1D parameter texture revealing node relocation caused by relaxation.

ABSTRACT

Virtual prototyping and numerical simulations are increasingly replacing real mock-ups and experiments in industrial product development. Many of these simulations, e.g. for the purpose of crash worthiness explorations, are based on Finite Element Analysis (FEA). In order to accelerate the development cycle, simulation engineers want to be able to modify their FE models without going back to the CAD department and without remeshing. Currently, there are no intuitive tools available that offer the possibility of modification and processing of FE components while maintaining the properties relevant to the simulation. In this paper we present interactive algorithms for intuitive, fast, and robust editing of FE models and appropriate visualization techniques to support engineers in understanding these models. New kinds of manipulators and feedback mechanisms enable easy manipulation of FE models. To ensure a good quality of the deformed mesh we use relaxation extended by algorithms preserving the features of the FE surface. For areas of large deformation we provide interactive methods to perform a remeshing in situ.

Keywords: freeform modeling, interaction, smoothing, optimization, manipulators

1. INTRODUCTION

Computer aided design (CAD) has evolved to an irreplaceable tool in the daily work routine of a design engineer, and it has definitely sped up the development cycle. The adoption of computer technology in this field is also useful when the design is transfered to the final production stage, e.g. by steering computerized numerical control (CNC) machines. However, the development cycle has been decelerated for a long time by the need for experiments to prove the structural and (aero-) dynamic performance of the designed model. To carry out these experiments, many expensive prototypes used to be built in a time-consuming process. During the last two decades this procedure has changed and an increasing number of test runs with real prototypes is being replaced by virtual simulations. Thanks to the growing processing power of modern parallel computers and to efficient algorithms it is now possible to calculate complex non-linear and highly dynamic processes like crash worthiness simulations within two or three days. In general, the analytical surfaces must be converted into an FE mesh for such a simulation. For this purpose, many conversion algorithms (e.g. [1, 2]) have been

developed, but they are far from being perfect and most of them are tailored to a specific kind of simulation or preservation of a particular model property. Therefore, a lot of expert knowledge and manual work is still required to achieve a high quality mesh. Some years ago, an improvement in the numerical algorithms was introduced, which alleviated this task by supporting individually meshed components instead of a single consistent mesh for a whole car. This made it possible to exchange only some parts without having to remesh the complete car model. Additionally, the assembly of the different car components can be reproduced more realistically by this approach due to the introduction of spotwelds or adhesive bondings. This change in the computational workflow combined with other efficiency gains results in a huge acceleration of the complete development cycle and even allows for stochastic analysis of model variants.

The traditional workflow changes by introducing this new approach. The engineers do not any longer need fast and reliable tools for meshing a complete car body at once and they do not have to care about congruent nodes to connect miscellaneous parts. Instead, they need appropriate tools to create interconnections between the various car components in a fast and easy way. These interconnections—e.g. spotwelds—are usually placed at flanges, i.e. where the surfaces of two or more components run parallel at a very small distance. Sometimes, adjacent materials may penetrate or perforate each other at such flange areas, because of sampling discrepancies in the discretization. In general, such meshing inconsistencies can be detected automatically and our preprocessing tool highlights affected regions with a color textures corresponding to the degree of inconsistency. As long as the penetrating area is well-defined, i.e. the affected parts are not folding or interfering in a complex manner, the perforation can be solved automatically. However, often manual modification is needed. Usually this means that the engineer has to ask the CAD department for corrected components, which have to be remeshed from scratch. This remeshing might reintroduce problems that arise from imperfect sampling in the meshing algorithms. Therefore it can be very profitable to have a comfortable editor that is able to manipulate FE meshes directly. Such an editor also offers the possibility to create variants of existing components very fast—e.g. the integration of crimpings for structural stiffening. In general these modification operations are focused on a certain region of a component and therefore the editing tool has to provide methods the engineer can use to select the three-dimensional surface region he wants to interact with. When performing an editing operation some of the finite elements might be deformed a lot. However, the elements must meet certain shape criteria to guarantee reasonable simulation results. Therefore our tool detects violations of these criteria and uses special glyphs to highlight the concerned elements. Once the modification is completed intelligent algorithms try to straighten out these errors without destroying the properties and features of the part. This mesh smoothing is not sufficient where strong deformation occurs, and we will present an approach to restructure the affected elements to fix these erroneous regions—comparable to a remeshing in situ. We will go into detail of the entire procedure in section 3 and 4. Current graphics hardware can support these preprocessing methods by using textures to highlight problematic regions or to compare various modeling states. The latter also can be used to compare the states before and after an automatic or manual modification.

Section 5 will show how the algorithms presented in this paper can be applied to a typical problem and it will close with some conclusions. In the following section we will discuss various previous work that provides the basis for our improvements.

2. RELATED WORK

Numerical simulations, such as crash worthiness explorations, work on FE meshes. So if the engineers want to interact with their computation model most applications can not be used because they are geared towards traditional design jobs. In another approach the engineers have the possibility to build complex constructions from simple geometric primitives, but neither of the approaches is developed to interact with FE models as they rise different needs in the tools: Other prerequisites such as the shape of the finite elements have to be provided and other fast and intuitive interaction and editing methods are needed—especially during preliminary design stages.

Therefore, we focused on intuitive interaction techniques, keeping them simple and laconic and requiring only one or two mouse clicks, a behavior which is also suggested by [3]. In [4] it is shown how perforations and penetrations of interfering parts can be removed automatically and how individually meshed components can be connected by interactively placing adhesive bondings or spotwelds along curved flanges. As mentioned above sometimes ambiguous tasks arise and so a fully automatic solution cannot always be provided. To manually edit such problematic areas [5] suggested directly manipulated free-form deformation. In this approach the final deformation is defined by spline volumes and interaction is steered by sliders and buttons. As this is not very intuitive, a better way is to use three-dimensional widgets the modification can be controlled in situ with. SGI's Inventor [6] demonstrates with its manipulators how to use widgets for 3D interaction. Various authors [7, 8, 9] presented 3D widgets that can be used for manipulation, but some of them are either technically immature or not suitable for FE modeling operations. A solution using simple and intuitive 3D glyphs for FE models was presented by [10], but it lacked support for direct manipulation.

Our recent approach to simple and at the same time versatile modification of FE meshes uses mouse drags as input and a variety of basis functions to describe the deformation of the surrounding surface. It is similar to the approach presented in [11], which was developed independently and which works on triangulated surfaces. For smoothing meshes, i.e.

after the modification step, various algorithms have been developed, most of them based on relaxation. The most common method is Laplacian smoothing, a simple recursive method where the nodes are directly adjusted depending on their adjacent nodes they have a common edge with. This method has been improved and extended by several authors to fit some special needs, e.g. [2, 1]. Another technique is the so-called optimization-based smoothing where the nodes are moved depending on the minimization of a special distortion metric. Also combinations of these techniques were presented, e.g. [12]. In [13, 14] a more detailed overview is given. Another approach is physically-based smoothing as presented e.g. in [15] where the edges in the mesh are represented by springs and the forces applied at the nodes depend on the ratio between the desired and the actual length of the edge. This approach is quite similar to the one we developed in this paper. As we have to deal with both quadrilaterals and triangles, we also have to take care of the diagonals in quadrilaterals. To avoid parallelograms, springs are added along the diagonals of quadrilaterals—unlike [15]—but their desired length is adapted to be $\sqrt{2}$ times longer than the edges of the element. Most smoothing and relaxation algorithms have been developed for smoothly curved or even closed surfaces, but in the case of car parts the surface is bounded and often contains holes or visible edges. Additionally, in order to preserve the significance of the numerical simulations, it is important that the readjusted nodes really lie on the surface originally modeled in CAD. This is why the most important improvement in our relaxation technique is that surface features (boundaries and visible edges) as well as the shape of the surface itself are detected and preserved.

In the following sections we show how we extended this knowledge by our own ideas to significantly improve the intuitive modification of FE meshes. By implementing our algorithms into the commercially available preprocessing application *scFEMod* [16] we have made this functionality available for productive use in the CAE departments of major German car manufacturers.

3. SELECTING AND EDITING IN 3D

The authors of [9] suggest editing operations by manipulating a skeleton line affecting the elements up to a certain distance from this line. This kind of selection during the modification process itself is rather limited. We decided to decouple the manipulation from the selection mechanism. In this way it is possible to select an arbitrary set of elements or nodes and perform various editing operations on them afterwards.

3.1 Selection Mechanisms

Selection mechanisms can be implemented in various ways. The simplest one is to provide a text box, where the user can enter the unique labels of the nodes. Surprisingly, some engineers know their datasets so well, that they prefer this cryptic method over anything else. Therefore, we also implemented such a dialog, but we extended it with boolean and range selection operations for improved convenience. However, with the widespread utilization of numerical simulation techniques also the number and the frequency of new models increases. Therefore, it gets more and more difficult to deal with a vast amount of different and continuously updated node IDs. Because of this, we suggest to use a much more intuitive method, usually utilized for selection in two-dimensional paint programs. There the user can drag the mouse, encircling the pixels he wants to select with a freehand curve and release the button to complete the action. In our application the engineer can draw a freehand line on the screen (see Fig. 2(a)), thereby cutting a sort of pyramidal frustum out of the 3D scene. To guarantee that the polygon is always closed, the start and end points are always connected (thin line in Fig. 2(a)). All nodes of the FE mesh that lie inside the pyramidal frustum are marked as selected and are highlighted via a white octahedron accordingly, as seen in Fig. 2(b). Hidden nodes are rendered transparent, so that at any time all selected nodes can be seen. It is unnecessary

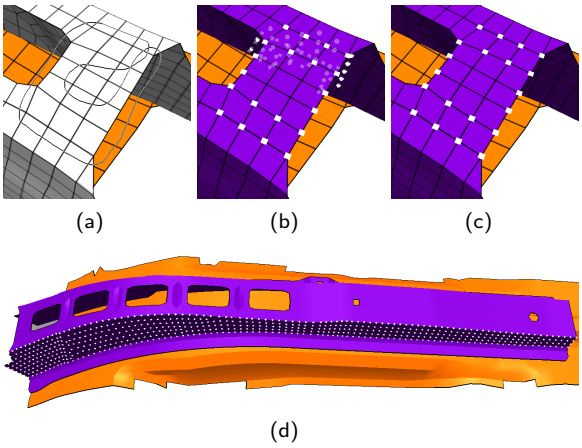


Figure 2: Freehand Selections: (a) encircling nodes; (b) selection without occlusion test, hidden nodes are rendered transparent; (c) selection originated from a subtraction operation; (d) selection of a region delimited by features.

to perform an expensive calculation in 3D to decide whether a node is outside or inside the freehand frustum. Instead, we project the node coordinates into the two-dimensional screen space and test the coordinates against the freehand outline. This problem can be solved very easily and fast using the point containment test presented in [17].

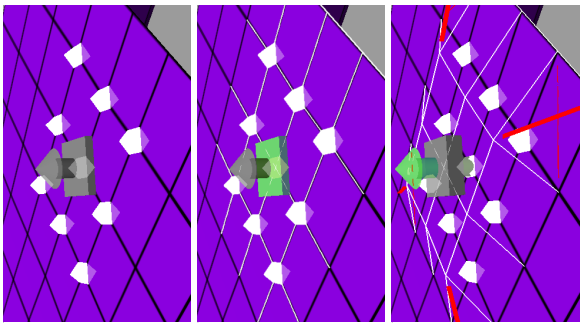
The user can choose whether he wants to select all the nodes inside the selection frustum of a component or if he only wants to select the currently visible nodes. Hidden nodes can be filtered out by shooting rays into the scene for every potential node candidate and checking for occlusion. This can be accelerated by rendering the car model with a unique color assigned to every element. We can then check the pix-

els in the neighborhood of the 2D coordinate of a potentially selected node for their colors or rather element labels. If none of these elements is adjacent to the node then it is occluded otherwise we can continue with the more precise ray intersection test. For greater convenience the user can add or subtract new selections to/from the existing ones, e.g. the selection in Fig. 2(c) has been achieved by selecting the node cluster with occlusion culling enabled and then deselecting the nodes in the center via subtraction.

Sometimes, the engineer wants to modify a complete segment of a component instead of a manually selected subset. Such a segment is delimited by features of the part, e.g. sharp edges as seen in Fig. 2(d). The user just has to click onto a part and all nodes belonging to the same region are selected at once. We use an approach presented in [18] to achieve a robust detection of the features of the FE mesh so that it is guaranteed that the algorithm does not choose bogus elements or nodes.

3.2 Mesh Modification

The simplest editing operation on a selection of nodes is a parallel translation. But in 3D even this cannot be accomplished as in 2D, where mouse movement can be mapped directly to a corresponding translation. Given 2D-only input, e.g. a mouse, 3D interaction with three or more DOF has to be split into multiple 2D movement actions. In our application we use the manipulator widget seen in Fig. 3(a). It consists of a disk and an arrow. The user can either click on the disk as in Fig. 3(b) and drag it around, whereby the displacements are constrained to the surface of the finite elements, or he can drag the arrow and shift the selection along the local surface normal like in Fig. 3(c). During interaction, the active part of the widget is highlighted in green and the modified surface is rendered as wire frame representation (white lines in Fig. 3(b) and 3(c)). The FE surface then is updated when the user releases the mouse button.



(a) (b) (c)

Figure 3: Manipulator for 3D movement: (a) neutral appearance when initially locked to node; (b) disk for movement on surface selected; (c) arrow for displacement along local normal selected and lifted by dragging with the mouse.

In most cases, simple parallel translations are not flexible enough. We implemented another approach that offers the possibility of much more complex deformations and which contains parallel translations as a special case. The interaction is kept as simple as before, but instead of moving all nodes along the same vector, we introduce a weighting function that describes the position of a selected node relative to the border of the selection and to the position where the user started dragging. If $d_{i,\text{border}}$ denotes the geodesic distance of a certain node i to the border of the selected region and $d_{i,\text{origin}}$ the geodesic distance of the node to the point where the user grabbed the surface, then the weight w_i can be calculated by

$$w_i = \frac{d_{i,\text{border}}}{d_{i,\text{origin}} + d_{\text{i,border}}} \,. \tag{1}$$

The range of values of all w_i is guaranteed to be within $[0,1]$. If we use these weights directly to scale the displacement vector we derive by the mouse movement along the widget arrow, then we obtain a deformation shaped like a cone or pyramid—depending on the shape of the selection. But the user can also choose from a variety of basis functions, e.g. when the engineer chooses a remapping of the weights similar to a Gaussian then he is able to achieve a result as shown in Fig. 4(a). Here, the Gaussian is flattened in the middle part, whereby the amount is configurable, and applied only into one direction. In the perpendicular direction the weights are ignored and set to constant 1 instead.

Rotations can be achieved by first defining the rotation axis and then describing an arc with the mouse which defines how much the selected region should be twisted (Fig. 4(b)). Furthermore, you can see that the rotation is performed to its whole extent at the free end without any weighting function applied. Here, free ends are areas between the axis of rotation and border segments where the boundary of the car component and of the selection match exactly, e.g. the left half of the component shown in Fig. 4(b). If the user wants to leave the free end fixed he can accomplish this by deselecting the outer row of nodes. This behavior in treating free endings applies not only to rotations but also to displacement operations and it is quite intuitive and enables many different kinds of modifications. For example it is very easy to elongate a certain section of a car component just by defining the section to be stretched by selecting its nodes and then dragging at the free border of the selection.

Each of these operations can be combined with an additional damping function that is applied to the border region of the selection, i.e. the area where the selected nodes flank unselected nodes. This damping term can be multiplied with the weighting function introduced in the last paragraph. Its goal is to achieve a smooth transition to the surrounding surface and to prevent the generation of badly shaped elements to a certain extent. This way it is possible to achieve the same deformation as presented in Fig. 4(a) by applying a parallel translation—i.e. the weighting function is set to constant 1—combined with an appropriate damping function. It turned out that such an extra damping function is more intu-

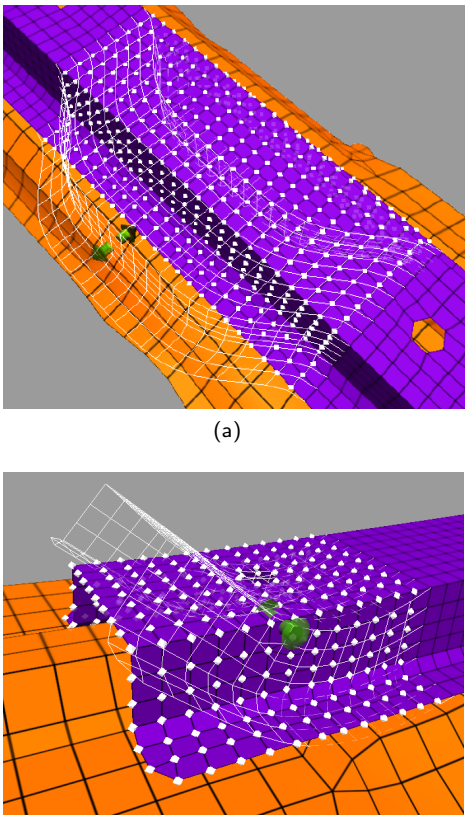


Figure 4: More complex modifications: (a) creating a bulge; (b) rotation with free ending.

itive and easier to handle than a vast pool of various mapping functions for the weighting term. The possibility to combine these two functions opens up a variety of complex deformations while keeping the two components rather simple and limiting them to a manageable number of parameters.

4. MESH OPTIMIZATION

Modification of a mesh can result in disadvantageous deformation of the finite elements. In general, it might lead to extreme angles between edges, or edges that vary too much within one element. Because of this the elements may become of poor quality for later computations. For FE methods "good elements" means uniform quadrilaterals and, where triangles cannot be avoided, equilateral triangles.

4.1 Visualization of Erroneous Elements

To give the engineer instantaneous feedback about the quality of the current FE mesh, the numerically relevant properties of each element are checked during the modification operation. Erroneous elements are highlighted immediately by using red glyphs which correspond to the error type. Critical elements are marked with yellow glyphs. An example for erroneous elements can be found in Fig. 3(c), where quadrilateral elements have been warped, i.e. where not all nodes of an element lie within the same plane. When warping occurs, the quadrilateral is bent around its two diagonals and we symbolize the larger of both bending angles by a thicker diagonal, respectively. Also problematic are edges that are too short because the minimum edge length of a crash dataset has a direct influence on the duration of the simulation time steps. If the edge length is chosen too small, shorter and therefore more simulation steps have to be calculated to achieve reliable results. If the engineer wants to limit the simulation effort this also limits the minimum edge length and edges violating this constraint are marked as shown in Fig. 5(a). The shape of finite elements is also important. Consequently, too small or too large angles are highlighted as can be seen in Fig. 5(b), 5(c) and 5(e). Another shape criterion is the ratio of the edge lengths of a finite element. This has to be evaluated only for quadrilaterals as for triangles it is checked implicitly via the angle criteria. Moderate irregularities still produce acceptable results in regions of low stress and are therefore marked only in light gray (yellow, Fig. 5(c)). An aspect ratio worse than a certain factor (usually ≈1:4) is marked in dark gray (red). Since triangles tend to produce less exact results than quadrilaterals, they should be avoided too, especially when they are clus-

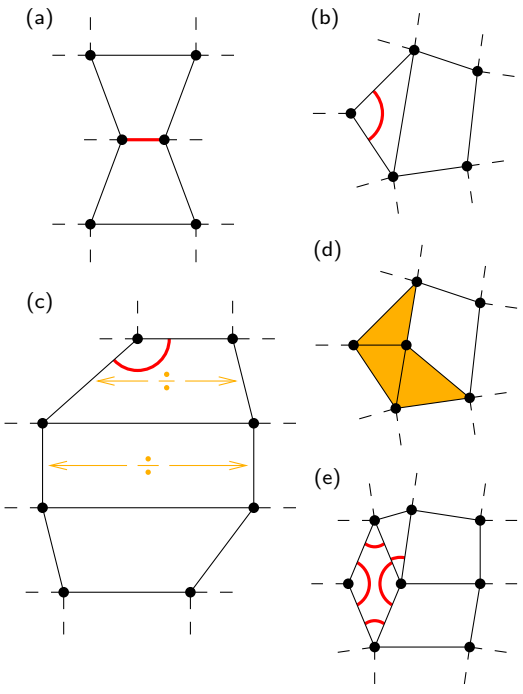


Figure 5: Various finite elements marked as erroneous: (a) edge too short; (b) angle too large (in triangle); (c) angle too large (in quadrilateral) and bad edge length ratios; (d) many adjacent triangles; (e) angles too small, in quadrilaterals this error is often combined with angles that are too large.

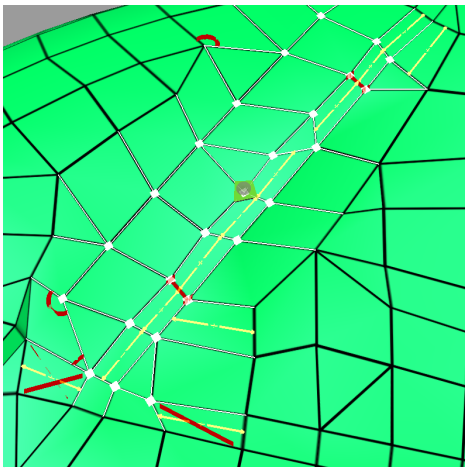


Figure 6: Examples of errors dynamically displayed while editing the mesh of a car component.

tered as in Fig. 5(d). Some of the discussed errors are presented in Fig. 6 on the FE surface of an authentic simulation model of a car component.

4.2 Mesh Relaxation

After editing the mesh, the nodes normally have to be readjusted, e.g. by relaxation, to make the mesh computable again. The relaxation model presented in this paper is based on a spring-mass model, where the edges of the mesh are represented by springs. To avoid parallelograms being classified as well shaped quadrilaterals additional springs are added along the diagonals of each quad in the mesh (see Fig. 7).

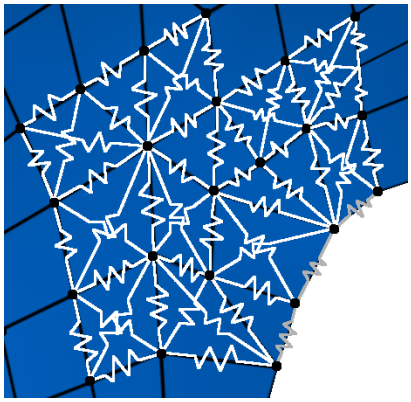


Figure 7: Spring-mass model for relaxation.

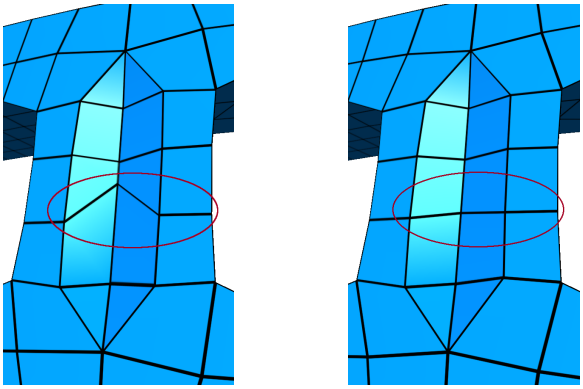
The rest length l_0 of the springs, and therefore the length of the edge represented by this spring, is set to the average length of the adjacent springs, where the diagonal ones are weighted with $1/\sqrt{2}$ according to the length of the diagonals in a square: let $n \in \mathbb{R}^3$ be the current node, l_0 is calculated by

$$l_0 = \frac{\sum \|n_{uc,i} - n\|/\sqrt{2} + \sum \|n_{c,j} - n\|}{\#n_{uc,i} + \#n_{c,j}}$$

where $n_{uc,i} \in \mathbb{R}^3$ are the adjacent nodes *not* connected to n (i.e. the diagonals in a quad), and $n_{c,i} \in \mathbb{R}^3$ are those connected to n by a common edge in the mesh. Accordingly $\#n_{uc,i}$ denotes the number of adjacent nodes not connected to n and $\#n_{uc,i}$ the number of nodes connected to n. Consequently, the force $F_s \in \mathbb{R}^3$ applied in this node is

$$F_s = \sum (n_{uc,i} - n) \left(||n_{uc,i} - n|| - l_0/\sqrt{2} \right) / ||n_{uc,i} - n|| + \sum (n_{c,j} - n) \left(||n_{c,j} - n|| - l_0 \right) / ||n_{c,j} - n|| .$$

Assuming the nodes have unit mass we get the ordinary differential equation (ODE) of second order $\ddot{\xi} = c_s F_s - \gamma \dot{\xi}$, where c_s is the spring constant, γ the damping constant, and $\xi \in \mathbb{R}^3$ the node coordinates. This equation can be rewritten as a system of 2 ODEs for computation. Just solving this system would lead to well shaped elements, but the features of the underlying surface would be mostly lost, i.e. the bordering curve would change as well as edges within the surface, and the surface itself would also get flattened. To avoid this we introduced some control mechanisms: after calculating the new position of each node by solving the ODE system we check if these coordinates are on the original surface or not. As we do not have the parametric representation of the surface but only the mesh, we interpolate the original surface and the surface feature lines onto which we project the nodes. Inner nodes belonging to a continuously curved part of the surface get projected onto the original surface interpolated by the coordinates of the neighbors in the last step. If a node is on a continuously curved surface or not is decided by calculating the angle between the normals of the adjacent elements adjoining in this node. If one or more of these angles differ too much from π, then the neighboring edges "in line" with the processed one are checked too, and if at least one of them also matches this criterion the feature line resulting from these edges is classified as a visible edge of the surface which is designated to be preserved. The described crease angle can be changed in the user interface. The detected feature lines are then computed according to the interpolation of the surface using the old coordinates of the adjacent nodes involved and the current node is moved constrained to this visible edge (see Fig. 8) by projecting the newly computed node position onto this curve. This procedure also preserves the bordering curve. Nodes where two or even more of such feature lines adjoin are classified to be corners that do not get moved at all. With this method the characteristic features of the surface concerning later computations stay unchanged. To get a visual feedback of that fact, the Euclidean distance between the originally modeled surface and the surface defined by the relaxed mesh is depicted in Fig. 11(c): distances smaller than 0.05mm are mapped to a transparent texture, distances between 0.05mm and 1mm are mapped to a color gradient beginning with dark (red) for 0.05mm up to light (yellow) for 1mm in the texture. Thus, as it can be seen in

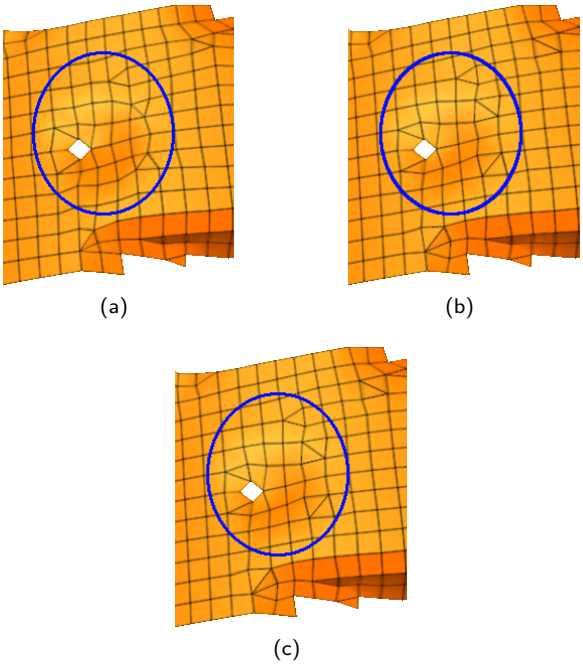


(a) (b)

Figure 8: Preservation of surface features: (a) original mesh, (b) relaxed mesh where the misplaced nodes were readjusted along the surface feature lines.

this figure, the shape of the surface is preserved very well, due to the implemented control mechanisms.

Since relaxing the mesh by solving the ODE system with these boundary conditions for one whole meshed car part with 500–1000 elements takes up to one or two minutes on a standard PC, we developed an algorithm that produces similar results but with interactive performance (Fig. 9): The mesh is relaxed hierarchically starting at a node selected by



(a) (b)

(c)

Figure 9: Comparing mesh relaxation methods: (a) original mesh; (b) hierarchically relaxed mesh; (c) non-hierarchically relaxed mesh.

the user e.g. in the region of highest irregularity. The displacement is calculated—similar to Laplacian smoothing—by adjusting each node "in the middle" of its neighbors—but with the same constraints on the node placement as explained in the method described above. In each iteration the nodes are moved starting with the selected node defining the center of the relaxation area. Then relaxation proceeds in circles around that center node by adjusting the adjacent nodes of those moved in the previous round and so on until no new node, in respect of the set of nodes moved in the current or the previous round, gets displaced further than a user-defined threshold thereby ending the current iteration step. In the next iteration step only nodes which were moved themselves and whose neighbors were displaced in the last iteration are checked for displacement again. With this procedure many nodes can be ignored in most of the iteration steps, and a lot of computing time can be saved. The disadvantage of this method is the fact that the mesh is not completely relaxed after the iterations and the method still moves nodes when called another time. But after the first or at least second time the new movements are very small. Comparing the results of both methods shows that even the first call to the hierarchical method already leads to acceptable results—see Fig. 9 and Fig. 11(d) where the non-hierarchical method solving the ODE system is compared to the result of the hierarchical method executed once. In Fig. 11(d) differences in the range from 1mm (dark, red) to 5mm (light, yellow) are mapped, whereas Fig. 11(b) depicts the total node displacement in the same range using the non-hierarchical method (see above).

In order to give more control to the engineers, it is also possible to restrict both relaxation methods to a group of selected nodes as depicted in Fig. 10. The user can mark nodes that shall be relaxed using the selection mechanisms described in section 3.1. Non-selected nodes do not get affected by the relaxation as can be seen in Fig. 10 where the selected nodes (white octahedrons) were moved, but all the others, such as those in the encircled area kept their position although the elements are very badly shaped.

To be able to evaluate the editing operations that have been performed, two kinds of visual feedback are provided: A simple one level undo function has been implemented, with a toggle for direct comparison. The other possibility is to start the application in a multiple view mode, where the user can have two windows with synchronized views. So the original part can be kept in one window while the corresponding part shown in the other window is being modified. This multiview mode can also be used to map geometrical differences between the two parts displayed via 1D textures (see Fig. 11). The user can decide whether he wants to map the node displacement caused by relaxation (Fig. 11(b) and Fig. 1(b)), or the Euclidean distance between the new node positions and the originally modeled surface (see Fig. 11(c)). In general, engineers do not want to cope with the displacement of single nodes and therefore the latter visualization method is the more significant one, as this one shows the deformation of the surface caused by the relaxation. This kind

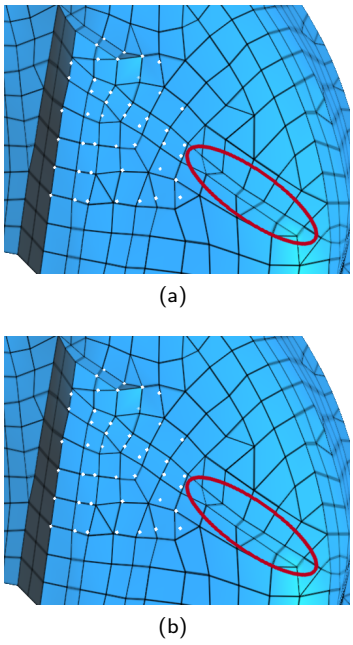


Figure 10: Relaxing only selected nodes: (a) node selection (white marked nodes); (b) relaxed region. Unselected nodes (encircled) are not adjusted.

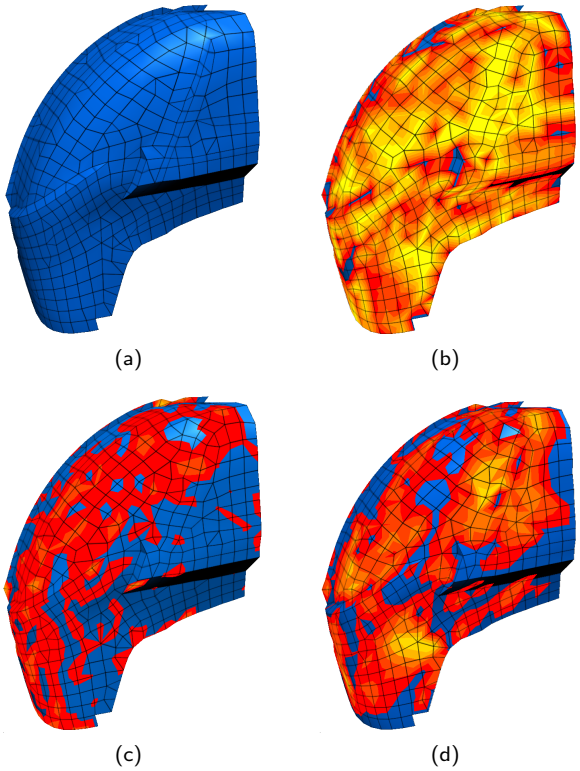


Figure 11: Comparing mesh relaxation methods: (a) original mesh; (b) node displacement from original to non-hierarchically relaxed mesh (< 1mm $\mapsto$ darkest (blue), > 5mm $\mapsto$ lightest (yellow)); (c) euclidean distance between non-hierarchically relaxed mesh and original (< 0.05mm $\mapsto$ darkest (blue), > 1mm $\mapsto$ lightest (yellow)); (d) difference between hierarchically and non-hierarchically relaxed mesh (< 1mm $\mapsto$ darkest (blue), > 5mm $\mapsto$ lightest (yellow)).

of texture mapping can also be used to visualize distances between arbitrary parts to detect e.g. penetrating flanges [4] as the range of distances to be mapped can be set by the user. The calculation is based on a bounding volume hierarchy [19] and therefore performs very well.

4.3 Mesh Restructuring

Relaxation is not always able to eliminate all errors in the mesh. Especially in cases in which a part gets elongated too much or the user creates a deep buckle, relaxation alone will not be able to produce a valid FE mesh. Nonetheless, relaxation still is a very powerful tool which will be also used as final polishing for the restructuring process which will be described in more detail in this section. The goal of restructuring is to perform small changes in the local topology of the mesh in a way that only erroneous elements and directly adjacent elements will be taken into account. It can be seen as a confined variant of remeshing, which guarantees to prevent unnecessary changes in the FE structure as a whole.

First, we determine the elements which were the worst ones before the relaxation procedure and try to fix them using one of the following operations: it is easy to repair warped quadrilaterals—introduced in section 4.1—by splitting each into two triangles along their "thick" bending diagonals. Short edges are collapsed to a common node placed at the center of the original edge as can be seen in Fig. 12(a). As a result, affected triangles will collapse to an edge and quadrilaterals will be converted into triangles. Elements containing large angles can be repaired by splitting them into two elements, whereas the newly formed edge contains the node with the large angle. A triangle is subdivided into two triangles, as depicted in Fig. 12(b). Quadrilaterals are either divided into a triangle and a quadrilateral or they are split into two quadrilaterals (see Fig. 12(c)). The latter method is preferred when the original quadrilateral has a bad aspect ratio. Additionally the algorithm has to take care of adjacent elements and check if they must be subdivided, too, in order to prevent hanging nodes as shown in Fig. 12(b) and 12(c). The operation presented in Fig. 12(d) usually is one of the last steps and tries to conjoin triangles into quadrilaterals. This decreases the number of triangles that might be produced by previous repair operations.

Elements containing too small angles cannot be fixed in a direct way and it cannot be guaranteed that it is always possible to repair such an error. We suggest the strategy to divide all quadrilaterals containing too small angles into two triangles as depicted by the solid gray (green) line in Fig. 12(e). Ad-

ditionally we split all the elements that are directly adjacent to such an erroneous element. In the majority of cases the operation presented in Fig. 12(d) is able to find valid combinations of adjacent triangles and merges them in such a way that a different variation of quadrilaterals is composed (e.g. by removing the dashed line in Fig. 12(e)). In doing so it is assured that only quadrilaterals of acceptable quality will be formed.

Finally, the neighborhood of erroneous elements is relaxed once again. If the resulting mesh still is not sufficient for simulation the algorithm undoes the relaxation and continues the restructuring also on less critical elements. This procedure is repeated until an FE mesh valid for crash worthiness simulations evolves, and experience shows that the algorithm converges very fast.

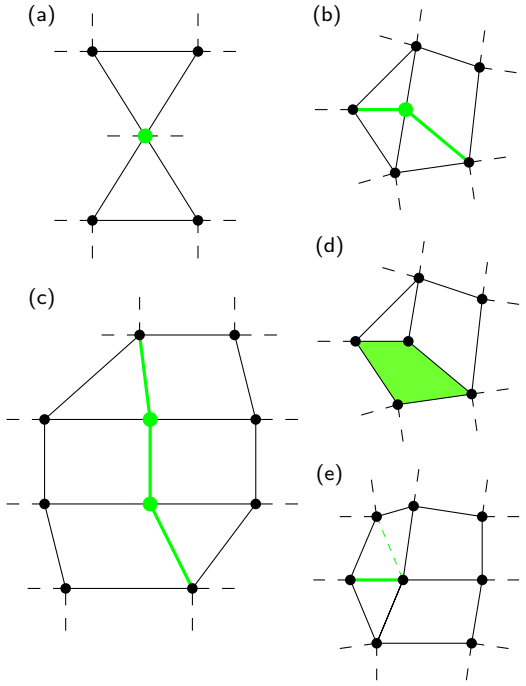


Figure 12: Finite elements from Fig. 5 have been fixed: (a) edge is collapsed to common node; (b) triangle and adjacent quadrilateral is divided; (c) stretched and adjacent elements are divided; (d) adjacent triangles are merged if possible; (e) quadrilateral is split and triangles are merged in a different way.

5. RESULTS AND CONCLUSIONS

Fig. 13 shows a simple, yet typical example for a problem which an engineer often faces. In the original model, seen in Fig. 13(a), the violet component perforates the orange part. Our application detects the erroneous area and visualizes it by marking the region with an alerting texture (Fig. 13(b)). Since it is a clearly defined perforation, it can be resolved automatically as shown in Fig. 13(c). In Fig. 13(d) it can

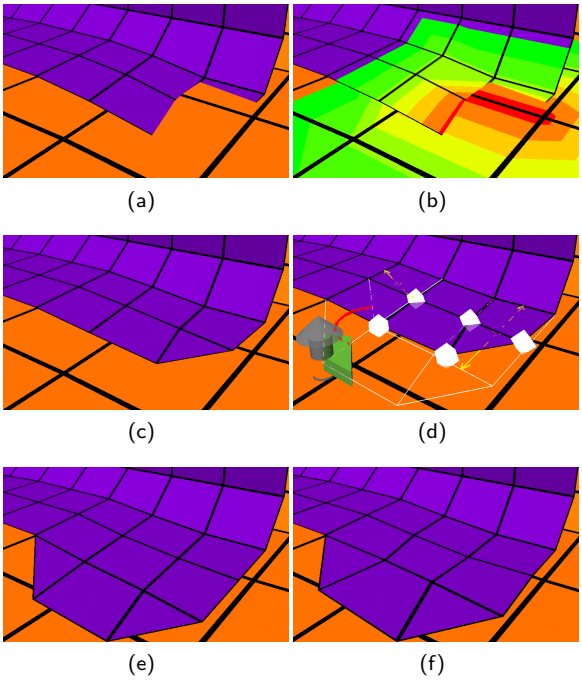


Figure 13: Various stages of repairing and editing a finite element model: (a) original mesh; (b) perforating parts are textured; (c) perforations and penetrations are removed; (d) user performs a modification and erroneous elements are marked interactively; (e) element errors are fixed automatically; (f) relaxation smoothes mesh whilst conserving features.

be seen how the engineer performs a stretching operation by dragging the 3D widget proposed in this paper. Erroneous elements are detected interactively and marked with appropriate glyphs, in this case two elements have a bad aspect ratio—yet not too critical for a valid numerical simulation—and one element contains an angle that is too large. Our preprocessing tool is able to repair these elements by locally restructuring the FE mesh and automatically inserting new elements (Fig. 13(e)) without the need for remeshing the whole part. This process is depicted in more detail in Fig. 14, in which we show how the subsequent repair operations are performed. Then the mesh is smoothed by using our relaxation approach, which guarantees that features—e.g. chamfers and sharp edges—are preserved. To compare the new mesh with the starting mesh one can use distance mapping as mentioned before.

We developed these methods in direct cooperation with engineers at the *BMW AG*. Most of our algorithms are no longer prototypes and have been transfered to the commercially available crash worthiness preprocessing tool *scFEMod*. Therefore, the techniques we presented in this paper are already being used widely by several German car manufacturers and their subcontractors. Due to their simplicity the presented methods have been quickly accepted and the engineers are now able to solve many mesh-related problems

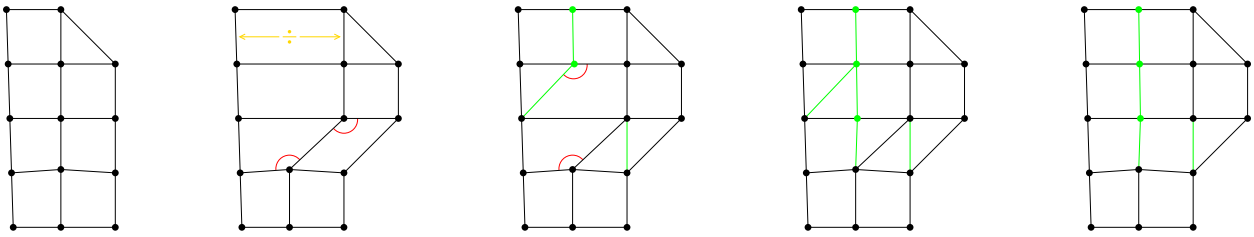


Figure 14: Locally restructuring an FE mesh: (a) original mesh; (b) modified mesh with erroneous elements; (c) remove large angle by splitting quadrilateral into two triangles, additionally subdivide quadrilateral with bad aspect ratio, also see Fig. 5(c) and 12(c); (d) remove remaining large angles (one is induced by the preceding operation); (e) merge adjacent triangles, also see Fig. 5(d) and 12(d).

on their own without the need for an additional loop through the CAD department. Generation of new variations of existing FE components—e.g. elongations or creation of stiffening corrugations and folds—is now possible using intuitive 3D widgets. Errors in the mesh that might appear during these editing operations are detected reliably and on-the-fly by our algorithms. Relaxation or local element restructuring can then be used to repair critical mesh regions. Although it is not possible to fix *all* kinds of meshing errors, our methods are nevertheless very powerful and successfully produce valid FE meshes in general. The texturing capabilities of standard graphics cards can be used to visualize the differences between various design stages, or to display erroneous regions of an FE mesh. This highlighting technique can be used also to prove that our methods for mesh repair and smoothing do not impair the surface and do preserve important features.

In this paper we demonstrated that there is a need for new modification methods for FE meshes and we presented solutions that can be operated intuitively. Glyphs and textures can be used to pinpoint erroneous or critical regions. Experience shows that our approach helps to vastly accelerate the development cycle in the automotive industry. We will continue our work on this field of research in close cooperation with the engineers, and we will investigate if it is reasonable to transfer our knowledge into other fields of engineering applications.

Acknowledgements

We would like to thank the *BMB+F* project *AutoOpt* for founding our research and the engineers at *BMW AG* for their cooperation within this project and for delivering insight into today's engineering problems. We would also like to thank Ove Sommer from *science+computing ag* and Reinhold Zöllner for providing many lines of code which are part of this work. Finally, we would like to thank Marcelo Magallón for fruitful discussion and many valuable hints.

References

[1] Blacker T.D., Stephenson M.B. "Paving: A new approach to automated quadrilateral mesh generation." *International Journal for Numerical Methods in Engineering*, vol. 32, 811–847, 1991

[2] Zhu J.Z., Zienkiewicz O.C., Hinton E., Wu J. "A new approach to the development of automatic quadrilateral mesh generation." *International Journal for Numerical Methods in Engineering*, vol. 32, 849–866, 1991

[3] Johnson J. *GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers*. Morgan Kaufmann Publishers, 2000

[4] Frisch N., Rose D., Sommer O., Ertl T. "Visualization and Pre-processing of Independent Finite Element Meshes for Car Crash Simulations." *The Visual Computer*, vol. 18, no. 4, 236–249, 2002

[5] Frisch N., Ertl T. "Deformation Of Finite Element Meshes Using Directly Manipulated Free-Form Deformation." *Proceedings of Seventh ACM Symposium on Solid Modeling and Applications 2002*, pp. 249–256. 2002

[6] Wernecke J. *The Inventor Mentor : Programming Object-Oriented 3D Graphics with Open Inventor, Release 2*. Addison-Wesley Pub Co, 1994

[7] Conner B.D., Snibbe S.S., Herndon K.P., Robbins D.C., Zeleznik R.C., van Dam A. "Three-dimensional widgets." *Proceedings of the 1992 symposium on Interactive 3D graphics*, pp. 183–188. ACM Press, 1992

[8] Döllner J., Hinrichs K. "Object-oriented 3D Modeling, Animation and Interaction." *The Journal of Visualization and Computer Animation*, vol. 8, no. 1, 33–64, 1997

[9] Grimm C., Pugmire D. "Visual Interfaces for Solids Modeling." *ACM Symposium on User Interface Software and Technology*, pp. 51–60. 1995

[10] Rose D., Frisch N., Ruehr T., Ertl T. “Interaktive Visualisierung neuer Elemente im virtuellen Automobil-Crashversuch.” *Tagungsband SimVis ’02, Magdeburg*. 2002

[11] Bendels G.H., Klein R., Schilling A. “Image and 3D-Object Editing with Precisely Specified Editing Regions.” *Workshop on Vision, Modelling, and Visualization VMV ’03*, pp. 451–460. Akademische Verlagsgesellschaft Aka GmbH, Berlin, November 2003

[12] Freitag L. “On combining Laplacian and optimization-based mesh smoothing techniques.” *AMD Trends in Unstructured Mesh Generation*, vol. 220, 37–43, 1997

[13] Amenta N., Bern M., Eppstein D. “Optimal point placement for mesh smoothing.” *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pp. 528–537. Society for Industrial and Applied Mathematics, 1997

[14] Canann S.A., Tristano J.R., Staten M.L. “An approach to combined Laplacian and optimization-based smoothing for triangular, quadrilateral, and quad-dominant meshes.” *Proceedings of 7th International Meshing Roundtable*, pp. 479–494. Sandia National Laboratories, 1998

[15] Löhner R., Morgan K., Zienkiewicz O.C. “Adaptive Grid Refinement for the Compressible Euler Equations.” *Accurency Estimates and Adaptive Refinements in Finite Element Computations*, pp. 281–297, 1986

[16] science + computing ag. “Efficient Preprocessing Using scFEMod.”, 2004. URL `http://www.science-computing.de/en/software/scfemod.html`

[17] Mortenson M.E. *Computer Graphics Handbook: Geometry and Mathematics*. Industrial Press, Inc., 1990

[18] Rose D., Ertl T. “Interactive Visualization of Large Finite Element Models.” *Workshop on Vision, Modelling, and Visualization VMV ’03*, pp. 585–592. Akademische Verlagsgesellschaft Aka GmbH, Berlin, 2003

[19] Gottschalk S., Lin M.C., Manocha D. “OBBTree: a hierarchical structure for rapid interference detection.” *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180. ACM Press, 1996